数理解析研究所講究録
1106 巻 1999 年 37-49

Domain-Free $\lambda\mu$-Calculus for Polymorphism and Call-by-Value*

Ken-etsu Fujita (藤田 憲悦)
Kyushu Institute of Technology, Iizuka 820-8502, Japan,
fujiken@dumbo.ai.kyutech.ac.jp

Abstract

We introduce a domain-free $\lambda\mu$-calculus of call-by-value as a short-hand for the second order Church-style. Our motivation comes from the observation that in Curry-style polymorphic calculi, control operators such as `callcc` or μ-operators cannot, in general, treat the terms placed on the control operator's left. Following the continuation semantics, we also discuss the notion of values in classical system, and propose an extended form of values. It is shown that the CPS-translation is sound with respect to domain-free $\lambda 2$ (2nd-order λ-calculus).

1 Introduction

On the basis of the Curry-Howard-De Bruijn isomorphism [How80], proof reductions can be regarded as computational rules, and the algorithmic contents of proofs can be used to obtain correct programs that satisfy logical specifications. The computational meaning of proofs has been investigated in a wide range of fields, including not only intuitionistic logic but also classical logic and modal logic [Koba97]. In the area of classical logic, there have been a number of noteworthy investigations including Griffin[Grif90], Murthy[Murt91], Parigot[Pari92], Berardi&Barbanera[BB93], Rehof&Sørensen[RS94], de Groote[Groo95] and Ong[Ong96]. As far as we know, however, polymorphic call-by-value calculus is less studied from the viewpoint of classical logic. In this paper, we introduce a domain-free $\lambda\mu$-calculus of call-by-value as a short-hand for the second order Church-style. Our motivation comes from the observation that in Curry-style polymorphic calculi, control operators such as `callcc` or μ-operators cannot, in general, treat the terms placed on the control operator's left. Following the continuation semantics, we also discuss the notion of values in classical system, and propose an extended form of values. It is shown that the CPS-translation is sound with respect to domain-free $\lambda 2$ (System F of Girard, Polymorphic calculus of Reynolds). We observe that the inverse of the soundness does not hold, and that adding $\perp$-reduction in Ong&Stewart [OS97] breaks down the soundness of the CPS-translation. As one of by-products, it can be obtained that the second order call-by-value $\lambda\mu$-calculus in domain-free style has the strong normalization property.

*This is a revised version of *Explicitly Typed $\lambda\mu$-Calculus for Polymorphism and Call-by-Value* presented at the 4th International Conference *Typed Lambda Calculi and Applications* (TLCA '99), L'Aquila, Italy, April 1999.

2 Style of (Typed) λ-Terms; Curry-Style, Church-Style, and Domain-free

There are well-known two style of typed lambda calculi, i.e., Curry-style and Church-style. Those styles are also called implicitly typed and explicitly typed, respectively. With respect to the simply typed lambda calculus $\lambda^{\rightarrow}$, there is a forgetful map from $\lambda^{\rightarrow}$ à la Church to à la Curry, and conversely, well-typed terms in $\lambda^{\rightarrow}$-Curry can be lifted to well-typed terms in $\lambda^{\rightarrow}$-Church [Bare92]. In the case of ML [Mil78], there also exists implicitly typed and explicitly typed systems, and they are essentially equivalent [HM93]. Hence, the implicitly typed system serves as a short-hand for the explicitly typed system. However, the equivalence between Curry-style and Church-style does not always hold for complex systems. Parigot [Pari92] introduced $\lambda\mu$-calculus in Curry-style as second order classical logic although $\lambda\mu$-calculus à la Church was also given [Pari97]. An intrinsically classical reduction is called the structural reduction that is a kind of permutative proof reductions in Prawitz [Praw71] or the so-called commutative cut. The $\lambda\mu$-calculus of Parigot is now known as a call-by-name system. If we construct a call-by-value $\lambda\mu$-calculus, then the Curry-style cannot work for a consistent system. In a call-by-value system of $\lambda\mu$, we can adopt a certain permutative reduction [Pari92, OS97], called the symmetric structural reduction, to manage the terms placed on the μ-operator's left. However, the symmetric structural proof reduction, in general, violates the subject reduction property in the Curry-style. Consider the following figure in which erasing type information of polymorphic terms makes it possible to apply the symmetric structural reduction uncorrectly:

$$
\dfrac{V : (\forall t.\sigma_1) \rightarrow \sigma_2 \qquad \dfrac{\dfrac{\dfrac{\begin{array}{c}\dfrac{M_1 : \sigma_1}{[\alpha]M_1 : \bot;\sigma_1^{\alpha}}\\ \vdots\end{array}}{M : \bot;\sigma_1^{\alpha}}}{\mu\alpha.M : \sigma_1}}{\mu\alpha.M : \forall t.\sigma_1}}{V(\mu\alpha.M) : \sigma_2}
\qquad \rhd \qquad
\dfrac{\begin{array}{c}\dfrac{\dfrac{V : (\forall t.\sigma_1) \rightarrow \sigma_2 \qquad \dfrac{M_1 : \sigma_1}{M_1 : \forall t.\sigma_1}\,(\forall I)^*}{VM_1 : \sigma_2}}{[\alpha](VM_1) : \bot;\sigma_2^{\alpha}}\\ \vdots \\ M[V \Rightarrow \alpha] : \bot;\sigma_2^{\alpha}\end{array}}{\mu\alpha.M[V \Rightarrow \alpha] : \sigma_2}
$$

where $M[V \Rightarrow \alpha]$ denotes a term obtained by replacing each subterm of the form $[\alpha]N$ in M with $[\alpha](VN)$. Here, when M is in the form of $[\alpha](\lambda x_1 \cdots x_n.M')$ and the type σ_1 depends on type of some x_i $(1 \leq i \leq n)$, the eigenvariable condition of $(\forall I)^*$ is broken down. For instance,

$$\lambda x.(\lambda f.(\lambda x_1 x_2.x_2)(fx)(f(\lambda x.x)))\ (\mu\alpha.[\alpha](\lambda y.\mu\beta.[\alpha](\lambda v.y)))$$

has type $t \rightarrow t \rightarrow t$. But this term is reduced to $\lambda x.x$ by the use of the symmetric structural reduction. Let $P \equiv \lambda f.(\lambda x_1 x_2.x_2)(fx)(f(\lambda x.x))$ and $Q \equiv \mu\alpha.[\alpha](\lambda y.\mu\beta.[\alpha](\lambda v.y))$. Then similarly

$$\lambda g.(\lambda x.g(PQx))(\lambda x.g(PQx)) : (\forall t'.(t' \rightarrow t')) \rightarrow t \rightarrow t$$

is reduced to $\lambda g.(\lambda x.g(xx))(\lambda x.g(xx))$. On the other hand, the case $\mu\alpha.M$ of $\mu\alpha.[\alpha](\lambda v.\mu\beta.[\alpha](\lambda x$ is a special case where the symmetric structural reduction is applicable even to polymorphic $\mu\alpha.M$, and then, for example,

$$\lambda x.((\lambda f.(\lambda x_1 x_2.x_2)(fx)(f(\lambda x.x)))\ (\mu\alpha.[\alpha](\lambda v.\mu\beta.[\alpha](\lambda x.x)))\ x) : t \rightarrow t$$

is reduced to $\lambda x.x$. This kind of phenomenon was first discovered by Harper & Lillibridge [HL91] as a counterexample for ML with `callcc`. From the viewpoint of classical proof reductions, the fatal defect can be explained such that in $\lambda\mu$-calculus à la Curry (2nd-order classical logic), an application of the symmetric structural reduction, in general, breaks down the eigenvariable condition of polymorphic generalization, and then the terms placed on the polymorphic μ-operator's left cannot be managed by the symmetric structural reduction. In terms of explicit polymorphism, in other words, an evaluation under Λ-abstractions cannot be allowed without restricting $\Lambda t.M$ to $\Lambda t.V$ [HL93a]. Even in the Damas-Milner style [DM82] (implicitly typed ML) plus control operators, a similar defect still happens under a ML-like call-by-value [HL93a, HL93b]. To avoid such a problem in implicitly typed ML with control operators, one can adopt an η-like expansion for polymorphic control operators [Fuji98], such that

$$\texttt{let}\ f = \mu\alpha.M_1\ \texttt{in}\ M_2\ \triangleright\ \texttt{let}\ f = \lambda x.\mu\alpha.M_1[\alpha \Leftarrow x]\ \texttt{in}\ M_2,$$

where each subterm in the form of $[\alpha](\lambda y.w)$ in M_1 is replaced with $[\alpha](\lambda y.w)x$.

Another natural way to avoid the problem is to take a domain-free system introduced by Barthe&Sørensen [BS97], see the following table:

Style \ Order	finitely typed	2nd order typed
Church-style	$\lambda x{:}\sigma.M$	$\Lambda t.M,\quad M\sigma$
Domain-free	$\lambda x.M$	$\Lambda t.M,\quad M\sigma$
Curry-style	$\lambda x.M$	M

In the above example, the term Q is a polymorphic term, and this type becomes $\forall t.(t \to t)$. Here, the explicitly typed term as a form of a value, $V \equiv \Lambda t.Q$ is used for β_v-reductions, such that

$$\lambda x.(\lambda f.(\lambda x_1 x_2.x_2)(ftx)(f(t \to t)(\lambda x.x)))\ V : t \to t \to t$$

is now reduced to $\lambda vx.x$. In the next section, under the call-by-value strategy we introduce a domain-free $\lambda\mu$-calculus, which is regarded as a short-hand for the complete Church-style. To obtain the results in this paper, it is enough to consider a system such that $\Lambda t.M$ is represented simply as ΛM such as a lifting and $M\sigma$ as $M()$, and $(\Lambda M)()$ is reduced to M. A similar observation is given for let-polymorphism by name in Leroy [Lero93]. The annotations Λ and $()$ for polymorphic terms play a role of choosing an appropriate computation under call-by-value. However, from the viewpoint of logic, a domain-free $\lambda\mu$-calculus is considered here rather than such a simplified polymorphism using the annotations.

On the other hand, Harper&Lillibridge [HL93a] extensively studied explicit polymorphism and CPS-conversion for F_ω with `callcc`. The call-by-value system $\lambda_V\mu$ introduced in section 3 can be regarded as a meaningful simplification of the second order fragment of their system.

3 $\lambda_V\mu$-Calculus in Domain-Free Style

Following the observation in the previous section, we introduce a domain-free $\lambda\mu$-calculus of call-by-value for polymorphism. Terms in domain-free style have domain-free λ-abstraction [BS97].

The types σ are defined from type variables t and a type constant $\perp$. We have a set of term variables $x, y, z, \cdots$, and a set of names (that will be called continuation variables later) $\alpha, \beta, \cdots$. The type assumptions are defined as usual, and Δ is used for a set of name-indexed types. The terms M are defined as term variables, λ-abstractions, applications, μ-abstractions, or named terms. Since we have sorted variables, i.e., term variable x and type variable t, we have explicit distinction between terms and types, and then λ-abstraction is used for both term variable and type variable abstractions.

From a logical viewpoint, the typing rule $(\perp E)$ for $\mu\alpha.M$ is regarded as a classical inference rule such that infer $\Gamma, \neg\Delta \vdash \mu\alpha.M : \sigma$ from $\Gamma, \neg\Delta, \alpha : \neg\sigma \vdash M : \perp$. The typing rule $(\bot I)$ for $[\alpha]M$ can be considered as a special case of $\bot$-introduction by the use of $(\to E)$. On the basis of the continuation semantics in the next section, a name can be interpreted as a continuation variable. In the rule $(\bot I)$, the continuation variable α appears only in the function-position, but not in the argument-position. Here, the negative assumption $\alpha : \neg\sigma$ corresponding to σ^α of $(\bot I)$ can be discharged only by $(\bot E)$. This style of proofs consisting of the special case of $\bot$-introduction is called a regular proof in Andou [Ando95]. The notion of values is introduced below as an extended form; the class of values is closed under both value-substitutions induced by (β_v) and left and right context-replacements induced by $(\mu_{l,r})$, as defined later. The definition of the reduction rules is given below under call-by-value. In particular, the classical reductions $(\mu_{l,r,t})$ below can be explained as a logical permutative reduction in the sense of Prawitz [Praw71] and Andou [Ando95]. Here, in the reduction of $(\mu\alpha.M)N \rhd \mu\alpha.M[\alpha \Leftarrow N]$, since both type of $\mu\alpha.M$ and type of each subterm M' with the form $[\alpha]M'$ in M can be considered as members of the segments ending with the type of $\mu\alpha.M$, the application of $(\to E, \forall E)$ is shifted up to each occurrence M', and then $M[y \Leftarrow N]$ (each $[\alpha]M'$ is replaced with $[\alpha](M'N)$) is obtained. This reduction is also called a structural reduction in Parigot [Pari92]. On the other hand, since a term of the form $\mu\alpha.M$ is not regarded as a value, $(\lambda x.M_1)(\mu\alpha.M_2)$ will not be a β-contractum, but will be a contractum of (μ_l) below, which can be considered as a symmetric structural reduction. $FV(M)$ stands for the set of free variables in M, and $FN(M)$ for the set of free names in M.

$\lambda_V\mu$:
Types $\sigma ::= t \mid \bot \mid \sigma \to \sigma \mid \forall t.\sigma$
Type Assumptions $\Gamma ::= \langle\,\rangle \mid \Gamma, x:\sigma$ $\qquad\qquad \Delta ::= \langle\,\rangle \mid \Delta, \sigma^\alpha$
Terms $M ::= x \mid \lambda x.M \mid MM \mid \lambda t.M \mid M\sigma \mid \mu\alpha.M \mid [\alpha]M$
Type Assignment

$$\Gamma \vdash x : \Gamma(x); \Delta$$

$$\frac{\Gamma \vdash M_1 : \sigma_1 \to \sigma_2; \Delta \quad \Gamma \vdash M_2 : \sigma_1; \Delta}{\Gamma \vdash M_1 M_2 : \sigma_2; \Delta}(\to E) \qquad \frac{\Gamma, x:\sigma_1 \vdash M : \sigma_2; \Delta}{\Gamma \vdash \lambda x.M : \sigma_1 \to \sigma_2; \Delta}(\to I)$$

$$\frac{\Gamma \vdash M : \forall t.\sigma_1; \Delta}{\Gamma \vdash M\sigma_2 : \sigma_1[t := \sigma_2]; \Delta}(\forall E) \qquad \frac{\Gamma \vdash M : \sigma; \Delta}{\Gamma \vdash \lambda t.M : \forall t.\sigma; \Delta}(\forall I)^*$$

$$\frac{\Gamma \vdash M : \sigma; \Delta}{\Gamma \vdash [\alpha]M : \bot; \Delta, \sigma^\alpha}(\bot I) \qquad \frac{\Gamma \vdash M : \bot; \Delta, \sigma^\alpha}{\Gamma \vdash \mu\alpha.M : \sigma; \Delta}(\bot E)$$

where $(\forall I)^*$ denotes the eigenvariable condition.
Values $V ::= x \mid \lambda x.M \mid \lambda t.M \mid [\alpha]M$
Term reductions
(β_v) $(\lambda x.M)V \rhd M[x := V]$; $\qquad (\eta_v)$ $\lambda x.Vx \rhd V$ if $x \notin FV(V)$;
(β_t) $(\lambda t.M)\sigma \rhd M[t := \sigma]$; $\qquad (\mu_t)$ $(\mu\alpha.M)\sigma \rhd \mu\alpha.M[\alpha \Leftarrow \sigma]$;
(μ_r) $(\mu\alpha.M_1)M_2 \rhd \mu\alpha.M_1[\alpha \Leftarrow M_2]$; $\qquad (\mu_l)$ $V(\mu\alpha.M) \rhd \mu\alpha.M[V \Rightarrow \alpha]$;
(rn) $[\alpha](\mu\beta.V) \rhd V[\beta := \alpha]$; $\qquad (\mu\text{-}\eta)$ $\mu\alpha.[\alpha]M \rhd M$ if $\alpha \notin FN(M)$,
where the term $M[\alpha \Leftarrow N]$ denotes a term obtained by M replacing each subterm of the form $[\alpha]M'$ in M with $[\alpha](M'N)$. That is, the terms (context) placed on $\mu\alpha.M$'s right is replaced in an argument position of M' in $[\alpha]M'$. In turn, the term $M[V \Rightarrow \alpha]$ denotes a term obtained by M replacing each subterm of the form $[\alpha]M'$ in M with $[\alpha](VM')$.

Values are reduced to simpler values by (η_v), eta-reduction and (rn), renaming rules, and those rules are restricted to values, whose condition is necessary to establish a sound CPS-translation in section 4. We note that as observed in Ong&Stewart [OS97], there are closed normal forms which are not values, called canonical forms, e.g., $\mu\alpha.[\alpha](\lambda x.\mu\beta.[\alpha](\lambda v.x))$. Those terms can be reduced by (S_3) in [Pari93] or ζ^{ext}_{fun} in [OS97], but in this case,

$(\mu\alpha.M)(\mu\beta.N)$ is reduced in the two ways (not confluent). Note also that the failure of operational extensionality for $\mu\mathrm{PCF}_v^-$ is demonstrated in [OS97]. In fact, ζ_{fun}^{ext} becomes admissible under the eta-reduction and (μ_r). Here, however a term in the form of $\mu\alpha.M$ is not a value, and we have the value-restricted (η_v) rather than the eta-reduction itself.

We denote $\triangleright_\mu$ by the one-step reduction induced by $\triangleright$. We write $=_\mu$ for the reflexive, symmetric, transitive closure of $\triangleright_\mu$. The notations such as $\triangleright_\beta$, $\triangleright_{\beta\eta}$, $\triangleright_\beta^+$, $\triangleright_{\beta\eta}^*$, $=_{\beta\eta}$, etc. are defined as usual, and $\triangleright_\beta^i$ denotes i-step β-reductions ($i \geq 0$).

Proposition 1 (Subject Reduction Property for $\lambda_V\mu$) *If we have $\Gamma \vdash M_1 : \sigma; \Delta$ and $M_1 \triangleright_\mu M_2$ in $\lambda_V\mu$, then $\Gamma \vdash M_2 : \sigma; \Delta$ in $\lambda_V\mu$.*

Proof. By induction on the derivation of $M_1 \triangleright_\mu M_2$. Note that in $\lambda_V\mu$, typing rules are uniquely determined depending on the shape of terms. □

The well-known type erasure M° is defined as follows:
$(x)^\circ = x$; $(\lambda x.M)^\circ = \lambda x.M^\circ$; $(M_1M_2)^\circ = M_1^\circ M_2^\circ$;
$(\lambda t.M)^\circ = M^\circ$; $(M\sigma)^\circ = M^\circ$; $(\mu\alpha.M)^\circ = \mu\alpha.M^\circ$; $([\alpha]M)^\circ = [\alpha]M^\circ$.
Then it can be seen that the typing relation is preserved between $\lambda_V\mu$ and implicitly typed $\lambda\mu$:
(i) If we have $\Gamma \vdash M : \sigma; \Delta$ in $\lambda_V\mu$, then $\Gamma \vdash M^\circ : \sigma; \Delta$ in implicit $\lambda\mu$.
(ii) If we have $\Gamma \vdash M_1 : \sigma; \Delta$ in implicit $\lambda\mu$, then there exists M_2 such that $M_1 = M_2^\circ$ and $\Gamma \vdash M_2 : \sigma; \Delta$ in $\lambda_V\mu$.
The set of types inhabited by terms coincides between implicit $\lambda\mu$ and $\lambda_V\mu$. However, erasing type information makes much more reductions possible, such as η-reduction of the erasure in Mitchell [Mit88], and the subject reduction property for M° is broken down, for example, a counterexample in section 2.

4 CPS-Translation for $\lambda_V\mu$-Calculus

To provide the CPS-translation, we define a domain-free $\lambda 2$ (see also [BS97]) as the intuitionistic fragment of $\lambda_V\mu$. Here, besides λ-variables $x, y, z, \cdots$ used in λ-calculus as usual, the system $\lambda 2$ has the distinguished variables $\alpha, \beta, \cdots$ called continuation variables. Reduction rules in domain-free $\lambda 2$ are also defined as usual under call-by-name. The term with the form $[\alpha]M$ (value) will be interpreted as $\lambda k.k(\overline{M}\alpha)$, where the representation of $\overline{M}\alpha$ is consumed by the continuation k, such as the case of λ-abstraction. The translation from $\lambda_V\mu$ to domain-free $\lambda 2$, with an auxiliary function Ψ for values, comes from Plotkin [Plot75].

Definition 1 (CPS-Translation) $\overline{x} = \lambda k.kx$; $\overline{\lambda x.M} = \lambda k.k(\lambda x.\overline{M})$;
$\overline{M_1M_2} = \lambda k.\overline{M_1}(\lambda m.\overline{M_2}(\lambda n.mnk))$; $\overline{\lambda t.M} = \lambda k.k(\lambda t.\overline{M})$;
$\overline{M\sigma} = \lambda k.\overline{M}(\lambda m.m\sigma^q k)$; $\overline{\mu\alpha.M} = \lambda\alpha.\overline{M}(\lambda x.x)$; $\overline{[\alpha]M} = \lambda k.k(\overline{M}\alpha)$.
$\Psi(x) = x$; $\Psi(\lambda x.M) = \lambda x.\overline{M}$; $\Psi(\lambda t.M) = \lambda t.\overline{M}$; $\Psi([\alpha]M) = \overline{M}\alpha$.
$t^q = t$; $(\sigma_1 \to \sigma_2)^q = \sigma_1^q \to \neg\neg\sigma_2^q$; $(\forall t.\sigma)^q = \forall t.\neg\neg\sigma^q$.

According to the continuation semantics of Meyer&Wand [MW85], our definition of the CPS-translation can be read as follows: If we have a variable x, then the value x is passed on to the continuation k. In the case of a λ-abstraction, a certain function that will take two arguments is passed on to the continuation k. If we have a term with a continuation variable α, then a certain function with the argument α is passed on to the continuation k, where the variable α will be substituted by a continuation. Here, it would be natural that a value is regarded as the term that is mapped by Ψ to some term consumed by the continuation k, since the continuation is the context in which a term is evaluated and then to which the value is sent. Our notion of values as an extended form is derived following this observation.

Lemma 1 *Let $=$ denote the definitional equality of the CPS-translation.*
(i) For any term M where $k \notin FV(M)$, $\lambda k.\overline{M}k \ \triangleright_\beta \overline{M}$.
(ii) For any value V, $\overline{V} = \lambda k.k\Psi(V)$.
(iii) For any term M, value V, and type σ, we have $\overline{M[x := V]} = \overline{M}[x := \Psi(V)]$ and $\overline{M[t := \sigma]} = \overline{M}[t := \sigma^q]$.

The above lemma can be proved by straightforward induction. On the basis of the CPS-translation, the left and right context-replacements $M[\alpha \Leftarrow M_1]$ and $M[V \Rightarrow \alpha]$ can be interpreted as the following substitutions for continuation variables, respectively.

Lemma 2 *Let M contain i free occurrences of $[\alpha]$ where $i \geq 0$. Then we have that $\overline{M[\alpha \Leftarrow M_1]} \ \triangleright_\beta^i \ \overline{M}[\alpha := \lambda m.\overline{M_1}(\lambda n.mn\alpha)]$ and $\overline{M[\alpha \Leftarrow \sigma]} \ \triangleright_\beta^i \ \overline{M}[\alpha := \lambda m.m\sigma^q\alpha]$.*

Proof. By induction on the structure of M. We show only the following case:
Case of $[\alpha]M$, where M contains i free occurrences of $[\alpha]$:
$\overline{([\alpha]M)[\alpha \Leftarrow M_1]} = \lambda k.k((\lambda k'.\overline{M[\alpha \Leftarrow M_1]}\lambda m.\overline{M_1}(\lambda n.mnk'))\alpha)$
$\triangleright_\beta \lambda k.k(\overline{M[\alpha \Leftarrow M_1]}\lambda m.\overline{M_1}(\lambda n.mn\alpha))$
$\triangleright_\beta^i \lambda k.k(\overline{M}[\alpha := \lambda m.\overline{M_1}(\lambda n.mn\alpha)](\lambda m.\overline{M_1}(\lambda n.mn\alpha)))$
$= \overline{[\alpha]M}[\alpha := \lambda m.\overline{M_1}(\lambda n.mn\alpha)]$. □

Lemma 3 *For any term M and value V, $\overline{M[V \Rightarrow \alpha]} \ \triangleright_\beta^{3i} \ \overline{M}[\alpha := \lambda n.\Psi(V)n\alpha]$, where M contains i free occurrences of $[\alpha]$.*

Proof. By induction on the structure of M. Only the case of $[\alpha]M$ is shown, where M contains i-occurrences of $[\alpha]$:
$\overline{([\alpha]M)[V \Rightarrow \alpha]} = \lambda k.k((\lambda k'.\overline{V}(\lambda m.\overline{M[V \Rightarrow \alpha]}(\lambda n.mnk')))\alpha)$
$\triangleright_\beta \lambda k.k((\lambda k'.k'\Psi(V))(\lambda m.\overline{M[V \Rightarrow \alpha]}(\lambda n.mn\alpha)))$
$\triangleright_\beta^2 \lambda k.k(\overline{M[V \Rightarrow \alpha]}(\lambda n.\Psi(V)n\alpha)) \triangleright_\beta^{3i} \lambda k.k(\overline{M}[\alpha := \lambda n.\Psi(V)n\alpha](\lambda n.\Psi(V)n\alpha))$
$= \overline{[\alpha]M}[\alpha := \lambda n.\Psi(V)n\alpha]$. □

Lemma 4 *If we have $M \triangleright_\mu N$ in $\lambda_V\mu$, then $\overline{M} =_{\beta\eta} \overline{N}$ in domain-free $\lambda 2$.*

Proof. By induction on the derivation of $M \triangleright_\mu N$. We show some of the cases:

(β_v) $(\lambda x.M)V \triangleright M[x := V]$:
$\overline{(\lambda x.M)V} = \lambda k_1.(\lambda k_2.k_2(\lambda x.\overline{M}))(\lambda m.\overline{V}(\lambda n.mnk_1))$
$\triangleright_\beta^2 \lambda k_1.\overline{V}(\lambda n.(\lambda x.\overline{M})nk_1)$
$\triangleright_\beta \lambda k_1.\overline{V}(\lambda x.\overline{M}k_1) = \lambda k_1.(\lambda k.k\Psi(V))(\lambda x.\overline{M}k_1)$
$\triangleright_\beta^2 \lambda k_1.\overline{M}[x := \Psi(V)]k_1 = \lambda k_1.\overline{M[x := V]}k_1 \triangleright_\beta \overline{M[x := V]}$.
(η_v) $\lambda x.Vx \triangleright V$:
$\overline{\lambda x.Vx} = \lambda k.k(\lambda x.(\lambda k'.(\overline{V}(\lambda m.\overline{x}(\lambda n.mnk')))))$
$\triangleright_\beta^2 \lambda k.k(\lambda x.(\lambda k'.(\overline{V}(\lambda m.mxk')))) = \lambda k.k(\lambda x.(\lambda k'.(\lambda k''.k''\Psi(V))(\lambda m.mxk')))$
$\triangleright_\beta^2 \lambda k.k(\lambda x.(\lambda k'.\Psi(V)xk'))$
$\triangleright_\eta \lambda k.k(\lambda x.\Psi(V)x) \triangleright_\eta \lambda k.k\Psi(V) = \overline{V}$.

$\underline{\quad(\beta_t)\ (\lambda t.M)\sigma \rhd M}$:
$\overline{(\lambda t.M)\sigma} = \lambda k.(\lambda k'.k'(\lambda t.\overline{M}))(\lambda m.m\sigma^q k)$
$\rhd_\beta^2 \lambda k.(\lambda t.\overline{M})\sigma^q k \rhd_\beta \lambda k.\overline{M}[t := \sigma^q]k \rhd_\beta \overline{M[t := \sigma]}$.
$\underline{\quad(\mu_t)\ (\mu\alpha.M)\sigma \rhd \mu\alpha.M[\alpha \Leftarrow \sigma]}$:
$\overline{(\mu\alpha.M)\sigma} = \lambda k.(\lambda\alpha.\overline{M}(\lambda x.x))(\lambda m.m\sigma^q k)$
$\rhd_\beta \lambda\alpha.\overline{M}[\alpha := \lambda m.m\sigma^q\alpha](\lambda x.x) =_\beta \lambda\alpha.\overline{M[\alpha \Leftarrow \sigma]}(\lambda x.x) = \overline{\mu\alpha.M[\alpha \Leftarrow \sigma]}$.
$\underline{\quad(\mu_r)\ (\mu\alpha.M)N \rhd \mu\alpha.M[\alpha \Leftarrow N]}$:
$\overline{(\mu\alpha.M)N} = \lambda k.(\lambda\alpha.\overline{M}(\lambda x.x))(\lambda m.\overline{N}(\lambda n.mnk))$
$\rhd_\beta \lambda k.\overline{M}[\alpha := \lambda m.\overline{N}(\lambda n.mnk)](\lambda x.x) = \lambda\alpha.\overline{M}[\alpha := \lambda m.\overline{N}(\lambda n.mn\alpha)](\lambda x.x)$
$=_\beta \lambda\alpha.\overline{M[\alpha \Leftarrow N]}(\lambda x.x) = \overline{\mu\alpha.M[\alpha \Leftarrow N]}$.
$\underline{\quad(\mu_l)\ V(\mu\alpha M) \rhd \mu\alpha.M[V \Rightarrow \alpha]}$:
$\overline{V(\mu\alpha.M)} = \lambda k.\overline{V}(\lambda m.(\lambda\alpha.\overline{M}(\lambda x.x))(\lambda n.mnk))$
$\rhd_\beta \lambda k.(\lambda k'.k'\Psi(V))(\lambda m.\overline{M}[\alpha := \lambda n.mnk](\lambda x.x))$
$\rhd_\beta^2 \lambda k.\overline{M}[\alpha := \lambda n.\Psi(V)nk](\lambda x.x) = \lambda\alpha.\overline{M}[\alpha := \lambda n.\Psi(V)n\alpha](\lambda x.x)$
$=_\beta \lambda\alpha.\overline{M[V \Rightarrow \alpha]}(\lambda x.x) = \overline{\mu\alpha.M[V \Rightarrow \alpha]}$.
$\underline{\quad(rn)\ [\alpha](\mu\beta.V) \rhd V[\beta := \alpha]}$:
$\overline{[\alpha](\mu\beta.V)} = \lambda k.k((\lambda\beta.\overline{V}(\lambda x.x))\alpha)$
$\rhd_\beta \lambda k.k(\overline{V}[\beta := \alpha](\lambda x.x)) = \lambda k.k((\lambda k'.k'\Psi(V)[\beta := \alpha])(\lambda x.x))$
$\rhd_\beta^2 \lambda k.k\Psi(V)[\beta := \alpha] = \overline{V[\beta := \alpha]}$.
$\underline{\quad(\mu\text{-}\eta)\ \mu\alpha.[\alpha]M \rhd M}$:
$\overline{\mu\alpha.[\alpha]M} = \lambda\alpha.(\lambda k.k(\overline{M}\alpha))(\lambda x.x)$
$\rhd_\beta^2 \lambda\alpha.\overline{M}\alpha \rhd_\beta \overline{M}$. □

Now, we have confirmed the soundness of the translation in the sense that equivalent $\lambda_V\mu$-terms are translated into equivalent domain-free $\lambda 2$-terms. This property essentially holds for untyped terms.

Proposition 2 (Soundness of the CPS-Translation) *If we have $M =_\mu N$ in $\lambda_V\mu$, then then $\overline{M} =_{\beta\eta} \overline{N}$ in domain-free $\lambda 2$.*

The translation logically establishes the double negation translation of Kuroda. For a set of name-indexed formulae Δ, we define $(\sigma^\alpha, \Delta)^q$ as $\alpha : \neg\sigma^q, \Delta^q$.

Proposition 3 *If $\lambda_V\mu$ has $\Gamma \vdash M : \sigma; \Delta$, then $\lambda 2$ has $\Gamma^q, \Delta^q \vdash \overline{M} : \neg\neg\sigma^q$.*

Proof. By induction on the derivation. □

From the consistency of domain-free $\lambda 2$, it is derived that $\lambda_V\mu$ is consistent in the sense that there is no closed term M such that $\vdash M : \bot;$ in $\lambda_V\mu$.

With respect to Proposition 2, it is known that the implication is, in general, not reversible. The counterexample in [Plot75] is not well-typed. Even though we consider well-typed $\lambda_V\mu$-terms, the completeness does not hold for $\lambda_V\mu$: If we have $M_1 \equiv (\lambda x.x)(xy)$ and $M_2 \equiv xy$ in $\lambda_V\mu$, then $\overline{M_1} =_{\beta\eta} xy =_{\beta\eta} \overline{M_2}$ in $\lambda 2$, but $M_1 \neq_\mu M_2$ in $\lambda_V\mu$. Note that in this counterexample, if one excluded η-reduction, then $\overline{M_1} \neq_\beta \overline{M_2}$. Following Hofmann [Hof95], the rewriting rules of $\lambda_V\mu$ are weak from the viewpoint of the semantics, since `Ident`, $(\lambda x.x)M = M$ is necessary in this case.

According to Ong&Stewart [OS97], their call-by-value $\lambda\mu$-calculus has more reduction rules with the help of type annotation; $\bot$-reduction:

$V^{\bot\to\sigma}M^\bot \rhd \mu\beta^\sigma.M^\bot$ if $\sigma \not\equiv \bot$.

Here, assume that we have $N_1 \equiv (\lambda x.x)(x([\alpha]y))$ and $N_2 \equiv x([\alpha]y)$, such that $x : \bot \to \sigma, y : \sigma \vdash N_i : \sigma; \sigma^\alpha$ $(i = 1, 2)$ where $\sigma \not\equiv \bot$ in $\lambda_V\mu$. Then N_1 and N_2 are reduced to $N_3 \equiv \mu\beta.[\alpha]y$ by the use of $\bot$-reduction. Now, we have $\overline{N_1} =_{\beta\eta} x(\alpha y) =_{\beta\eta} \overline{N_2}$ in $\lambda 2$, but $\overline{N_3} =_\beta \lambda\beta.\alpha y$

in $\lambda 2$. This example means that the soundness of the CPS-translation is broken down for $\lambda_V\mu$ with $\perp$-reduction, even in the absence of η-reduction. However, on the basis of the correspondence between μ-operator and Felleisen's $\mathcal{C}$-operator [FFKD86] such that $\mu\alpha.M = \mathcal{C}(\lambda\alpha.M)$ and $[\alpha]M = \alpha M$, one obtains that $x(\alpha y) =_{\mathcal{C}} (\lambda x.\mathcal{A}(x))(\alpha y) =_{\mathcal{C}} \mathcal{A}(\alpha y) =_{\mathcal{C}} \mathcal{C}(\lambda\beta.\alpha y)$ in the equational theory $\lambda_{\mathcal{C}}$ [Hof95]. From the naive observation, Hofmann's categorical models for $\lambda_{\mathcal{C}}$ would also work for an equational version of call-by-value $\lambda\mu$-calculus.

Let $\triangleright_{\beta\eta r}$ be one-step $\triangleright_\mu$ consisting of (β_v), (β_t), (η_v), $(\mu\text{-}\eta)$, or (rn). Let $\triangleright_{st}$ be one-step $\triangleright_\mu$ consisting of (μ_l), (μ_r), or (μ_t). Following the proof of lemma 2, if $M_1 \triangleright_{\beta\eta r} M_2$, then $\overline{M_1} \triangleright^+_{\beta\eta} \overline{M_2}$. On the one hand, each $\triangleright_{st}$-step from M does not simply induce β-steps from $\overline{M}$, i.e., β-conversion may be used. To demonstrate the strong normalization for well-typed $\lambda_V\mu$-terms, it is enough to construct an infinite reduction path from $\overline{M}$ if M has an infinite reduction path. In the case of $\triangleright_{st}$, following lemmata 2 and 3, the CPS-translated terms without the β-conversion still have enough β-, η-redexes to construct an infinite reduction. For instance, in the case M_1 of $(V(\mu\alpha.M))N$, we have $M_1 \triangleright_{st} M_2 \triangleright_{st} M_3$, where $M_2 \equiv (\mu\alpha.M[V \Rightarrow \alpha])N$ and $M_3 \equiv \mu\alpha.M[V \Rightarrow \alpha][\alpha \Leftarrow N]$. Here, $\overline{M_1}$ can be reduced as follows:

$\overline{M_1} \triangleright^+_\beta N_2 \equiv \lambda k.(\lambda\alpha.\overline{M}id\theta_1)(\lambda m.\overline{N}(\lambda n.mnk)) \triangleright_\beta N_3 \equiv \lambda\alpha.\overline{M}id\theta_1\theta_2$,

where $id = \lambda x.x$, $\theta_1 = [\alpha := \lambda n.\Psi(V)n\alpha]$, and $\theta_2 = [\alpha := \lambda m.\overline{N}(\lambda n.mn\alpha)]$. We now have $\overline{M_2} \triangleright^*_\beta N_2$ and $\overline{M_3} \triangleright^*_\beta N_3$. Let $[N/\alpha]$ be either $[N \Rightarrow \alpha]$ or $[\alpha \Leftarrow N]$.

Lemma 5 *(i) If $M_1 \triangleright_{st} M_2 \triangleright_{st} M_3$, then $\overline{M_1} \triangleright^+_\beta N_2 \triangleright^+_\beta N_3$ for some $\lambda 2$-terms N_2 and N_3 such that $\overline{M_2} \triangleright^*_\beta N_2$ and $\overline{M_3} \triangleright^*_\beta N_3$.*
(ii) Let $\alpha \notin FN(N)$. If $M_1[N/\alpha] \triangleright_{\beta\eta r} M_2$, then $\overline{M_1}\theta_1 \triangleright^+_{\beta\eta} \overline{N_2}\theta_2$ for some $\lambda_V\mu$-term N_2 and substitutions θ_1 and θ_2 such that $\overline{M_1[N/\alpha]} \triangleright^*_\beta \overline{M_1}\theta_1$ and $\overline{M_2} \triangleright^*_\beta \overline{N_2}\theta_2$.

Proof. Let θ^γ_1 be $[\gamma := \lambda m.\overline{N}(\lambda n.mn\gamma)]$, $[\gamma := \lambda n.\Psi(V)n\gamma]$, or $[\gamma := \lambda m.m\sigma^q\gamma]$ for some N, V, and σ. Let $id = \lambda x.x$.
(i) To construct an infinite $\triangleright_\beta$-path from $\overline{M}$ if we have an infinite $\triangleright_{st}$-path from M, it is enough to verify that we have an infinite $\triangleright_\beta$-path from $\overline{M}$ in the case where one $\triangleright_{st}$-reduction induces a new $\triangleright_{st}$-redex. Therefore, by induction on the derivations of $\triangleright_{st}$ we show that if $M_1[N/z]\triangleright_{st}M_2$, then $\overline{M_1}\theta\triangleright^+_\beta N_2$ for some $\lambda 2$-term N_2 such that $\overline{M_1[N/z]}\triangleright^*_\beta\overline{M_1}\theta$ and $\overline{M_2} \triangleright^*_\beta N_2$.

$([\alpha](\mu\beta.M))[\alpha \Leftarrow N] \triangleright_{st} [\alpha](\mu\beta.M[\alpha \Leftarrow N][\beta \Leftarrow N])$:
$\overline{([\alpha](\mu\beta.M))[\alpha \Leftarrow N]} \triangleright^*_\beta \overline{[\alpha](\mu\beta.M)}\theta^\alpha_1$
$\triangleright_\beta \lambda k.k(\overline{M}id\theta^\alpha_1\theta^\beta_2) = \lambda k.k(\overline{M}id[\beta := \alpha]\theta^\alpha_1)$ where $\theta^\beta_2 = [\beta := \lambda m.\overline{N}(\lambda n.mn\alpha)]$.
$([\alpha](\mu\beta.M))[V \Rightarrow \alpha] \triangleright_{st} [\alpha](\mu\beta.M[V \Rightarrow \alpha][V \Rightarrow \beta])$:
$\overline{([\alpha](\mu\beta.M))[V \Rightarrow \alpha]} \triangleright^*_\beta \overline{[\alpha](\mu\beta.M)}\theta^\alpha_1$
$\triangleright_\beta \lambda k.k(\overline{M}id[\beta := \alpha]\theta^\alpha_1)$.
$([\alpha]V)[\alpha \Leftarrow \mu\beta.M] \triangleright_{st} [\alpha](\mu\beta.M[(V[\alpha \Leftarrow \mu\beta.M]) \Rightarrow \beta])$:
$\overline{([\alpha]V)[\alpha \Leftarrow \mu\beta.M]} \triangleright^*_\beta \overline{[\alpha]V}\theta^\alpha_1$
$\triangleright_\beta \lambda k.k((\lambda m.\overline{\mu\beta.M}(\lambda n.mn\alpha))(\Psi(V)\theta^\alpha_1)) \triangleright_\beta \lambda k.k(\overline{\mu\beta.M}(\lambda n.(\Psi(V)\theta^\alpha_1)n\alpha))$
$\triangleright_\beta \lambda k.k(\overline{M}id\theta^\beta_2)$ where $\theta^\beta_2 = [\beta := \lambda n.(\Psi(V)\theta^\alpha_1)n\alpha]$. Moreover, $\lambda k.k(\overline{M}id\theta^\beta_2) = \lambda k.k(\overline{M}id\theta^{\beta'}_2\theta^{\alpha'}_1)$ where $\theta^{\beta'}_2 = [\beta := \lambda n.\Psi(V')n\alpha]$, $V' = V[\alpha := \alpha']$, and $\theta^{\alpha'}_1 = [\alpha' := \lambda m.\overline{\mu\beta.M}(\lambda n.mn\alpha)]$.

In the above cases, the symbol $\lambda\beta$ in $\lambda k.k(\lambda\beta.\overline{M})$ obtained from $\overline{\mu\beta.M}$ is disappeared by executing $\triangleright_\beta$. However, this gives no defects to have an infinite $\triangleright_\beta$-path, since when the $\mu\beta.M$ corresponding to the disappeared $\lambda\beta$ performs infinite $\triangleright_{st}$, the arguments must be provided infinitely by other $\rhd_{st}$, and the infinite $\rhd_{st}$ induce infinite $\rhd_\beta$-steps. Moreover, the redex of renaming, $[\alpha](\mu\beta.M[N/z])$ can be simulated by $\rhd_\beta$-steps in N_2.

(ii) By induction on the derivation of $\rhd_{\beta\eta r}$. We show some of the cases.

(β_v) $([\alpha](\lambda x.M))[\alpha \Leftarrow V] \rhd_{\beta_v} [\alpha](M[\alpha \Leftarrow V][x := V])$:
$\overline{([\alpha](\lambda x.M))[\alpha \Leftarrow V]} \rhd^*_\beta \overline{[\alpha](\lambda x.M)}\theta^\alpha_1 = \lambda k.k((\lambda k'.k'(\lambda x.\overline{M}))\alpha)\theta^\alpha_1$
$\rhd_\beta \lambda k.k((\lambda m.\overline{V}(\lambda n.mn\alpha))(\lambda x.\overline{M}\theta^\alpha_1)) \rhd_\beta \lambda k.k(\overline{V}(\lambda n.(\lambda x.\overline{M}\theta^\alpha_1)n\alpha))$
$\rhd_\beta \lambda k.k(\overline{V}(\lambda x.\overline{M}\theta^\alpha_1\alpha)) \rhd^2_\beta \lambda k.k(\overline{M}[x := \Psi(V)]\theta^\alpha_1\alpha)$
$= \lambda k.k(\overline{M[x := V]}\theta^\alpha_1\alpha) = \lambda k.k(\overline{M'[x := V]}\theta^{\alpha'}_2\alpha) = \overline{[\alpha]M'[x := V]}\theta^{\alpha'}_2$ where $M' = M[\alpha := \alpha']$ and $\theta^{\alpha'}_2 = [\alpha' := \lambda m.\overline{V}(\lambda n.mn\alpha)]$.

(β_v) $([\alpha]V)[\lambda x.M \Rightarrow \alpha] \rhd_{\beta_v} [\alpha](M[x := (V[\lambda x.M \Rightarrow \alpha])])$:
$\overline{([\alpha]V)[\lambda x.M \Rightarrow \alpha]} \rhd^*_\beta \overline{[\alpha]V}\theta^\alpha_1 = \lambda k.k(\overline{V}\alpha)\theta^\alpha_1$
$\rhd_\beta \lambda k.k((\lambda n.\Psi(\lambda x.M)n\alpha)(\Psi(V)\theta^\alpha_1)) \rhd_\beta \lambda k.k(\Psi(\lambda x.M)(\Psi(V)\theta^\alpha_1)\alpha)$
$= \lambda k.k((\lambda x.\overline{M})(\Psi(V)\theta^\alpha_1)\alpha) \rhd_\beta \lambda k.k((\overline{M}[x := \Psi(V)]\theta^\alpha_1)\alpha)$
$= \lambda k.k((\overline{M[x := V]}\theta^\alpha_1)\alpha) = \overline{[\alpha]M[x := V']}\theta^{\alpha'}_2$ where $V' = V[\alpha := \alpha']$ and $\theta^{\alpha'}_2 = [\alpha' := \lambda n.\Psi(\lambda x.M)n\alpha]$.

(β_t) $([\alpha](\lambda t.M))[\alpha \Leftarrow \sigma] \rhd_{\beta_t} [\alpha](M[\alpha \Leftarrow \sigma][t := \sigma])$:
$\overline{([\alpha](\lambda t.M))[\alpha \Leftarrow \sigma]} \rhd^*_\beta \overline{[\alpha](\lambda t.M)}\theta^\alpha_1 = \lambda k.k((\lambda k'.k'(\lambda t.\overline{M}))\alpha)\theta^\alpha_1$
$\rhd_\beta \lambda k.k((\lambda m.m\sigma^q\alpha)(\lambda t.\overline{M}\theta^\alpha_1)) \rhd_\beta \lambda k.k((\lambda t.\overline{M}\theta^\alpha_1)\sigma^q\alpha)$
$\rhd_\beta \lambda k.k((\overline{M}[t := \sigma^q]\theta^\alpha_1)\alpha) = \lambda k.k((\overline{M[t := \sigma]}\theta^\alpha_1)\alpha)$
$= \overline{[\alpha]M'[t := \sigma]}\theta^{\alpha'}_2$ where $M' = M[\alpha := \alpha']$ and $\theta^{\alpha'}_2 = [\alpha' := \lambda m.\sigma^q\alpha]$.

(rn) $([\alpha](\mu\beta.V))[N/\gamma]$ where $\alpha \not\equiv \gamma$:
$\overline{([\alpha](\mu\beta.V))[N/\gamma]} \rhd^*_\beta \overline{[\alpha](\mu\beta.V)}\theta^\gamma_1 = \lambda k.k((\lambda\beta.\overline{V}id)\alpha)\theta^\gamma_1$
$\rhd^3_\beta \lambda k.k(\Psi(V)[\beta := \alpha])\alpha)\theta^\gamma_1 = \overline{V[\beta := \alpha]}\theta^\gamma_1$.

(η_v) $([\alpha](\lambda x.Vx))[N/\gamma]$ where $x \notin FV(V)$:
$\overline{([\alpha](\lambda x.Vx))[N/\gamma]} \rhd^*_\beta \overline{[\alpha](\lambda x.Vx)}\theta^\gamma_1 \rhd^+_{\beta\eta} \overline{[\alpha]V}\theta^\gamma_1$.

Moreover, we have that
$\Psi(\lambda x.Vx) \rhd^+_{\beta\eta} \Psi(V)$ and $\Psi([\alpha](\mu\beta.V)) \rhd^+_\beta \Psi(V[\beta := \alpha])$. □

Lemma 6 *If there exits an infinite $\rhd_\mu$-reduction path from $\lambda_V\mu$-term M, then $\overline{M}$ also has an infinite $\rhd_{\beta\eta}$-reduction path.*

Proof. From Lemma 5 and the proof of Lemma 4. □

From Proposition 3, Lemma 6 and the fact that domain-free $\lambda 2$ is strongly normalizing [BS97], the strong normalization property for $\lambda_V\mu$ can be obtained.

Proposition 4 (Strong Normalization Property for $\lambda_V\mu$) *Any well-typed $\lambda_V\mu$-term is strongly normalizable.*

It is observed [Fuji97] that the straightforward use of the Tait&Martin-Löf parallel reduction [Taka89] could not work for proving the Church-Rosser property for $\lambda\mu$ including renaming rule, contrary to the comments on Theorem 2.5 in [OS97]. Even though one defines parallel reduction $\gg$ as usual, we cannot establish that if $M_i \gg N_i$ $(i = 1, 2)$, then $M_1[\alpha \Leftarrow M_2] \gg N_1[\alpha \Leftarrow N_2]$; fact (iv) in the proof of Theorem 1 in [Pari92].

Lemma 7 (Weak Church-Rosser Property for $\lambda_V\mu$) *If $M \rhd_\mu M_1$ and $M \rhd_\mu M_2$, then $M_1 \rhd^*_\mu N$ and $M_2 \rhd^*_\mu N$ for some N.*

From Proposition 4 and Lemma 7, we can obtain the Church-Rosser property using Newman's lemma [Bare84].

Proposition 5 (Church-Rosser Theorem) *$\lambda_V\mu$ has the Church-Rosser property for well-typed terms.*

5 Comparison with Related Work and Concluding Remarks

We briefly compare $\lambda\mu_{ml}$ (ML+μ, see [Fuji99]) with ML [Mil78, DM82] together with `callcc` [HDM93]. In ML, the class of type variables is partitioned into two subclasses, i.e., the applicative and the imperative type variables. The type of `callcc` is declared with imperative type variables to guarantee the soundness of the type inference. On the basis of the classification, the typing rule for let-expression is given such that if the let-bound expression is not a value, then generalization is allowed only for applicative type variables; otherwise generalization is possible with no restriction. There is a simple translation from the ML-programs to the $\lambda\mu_{ml}$-terms, such that the two subclasses of type variables in ML are degenerated into a single class: $\lceil \mathtt{callcc}(M) \rceil = \mu\alpha.[\alpha](\lceil M \rceil(\lambda x.[\alpha]x))$;
$\lceil \mathtt{throw}\ M\ N \rceil = \mu\beta.\lceil M \rceil \lceil N \rceil$ where β is fresh.
However, there are some distinctions; according to Harper et al. [HDM93], the program:
$\mathtt{let}\ f = \mathtt{callcc}(\lambda k.\lambda x.\mathtt{throw}\ k\ (\lambda v.x))\ \mathtt{in}\ (\lambda x_1 x_2.x_2)(f\ \mathtt{1})(f\ \mathtt{true})$
is not typable in ML, since $\mathtt{callcc}(\lambda k.\lambda x.\mathtt{throw}\ k\ (\lambda v.x))$ with imperative type variables is not a value, and in the case of non-value expressions, polymorphism is allowed only for expressions with applicative type variables. If it were typable with `bool`, then this was reduced to `1` following the operational semantics. On the other hand, under the translation $\lceil\ \rceil$ together with type annotation, in $\lambda\mu_{eml}$ [Fuji99] we have
$\mathtt{let}\ f = \Lambda t.\mu\alpha.[\alpha]\lambda x.\mu\beta.[\alpha](\lambda v.x)\ \mathtt{in}\ (\lambda x_1 x_2.x_2)(f\ \mathtt{int}\ \mathtt{1})(f\ \mathtt{bool}\ \mathtt{true})$
with type `bool`, and this is now reduced to `true`, as in F_ω plus `callcc` under call-by-value, not under ML-like call-by-value [HL93a]. In turn, the following term
$\mathtt{let}\ f = \mu\alpha.[\alpha]\lambda x.\mu\beta.[\alpha](\lambda v.x)\ \mathtt{in}\ (\lambda x_1 x_2.x_2)(f\ \mathtt{1})(f\ \mathtt{2})$
with type `int` is reduced to `1` by the symmetric structural reduction. On the other hand, in $\lambda\mu_{iml}$ [Fuji99] we have
$\mathtt{let}\ f = \mu\alpha.[\alpha]\lambda x.\mu\beta.[\alpha](\lambda v.x)\ \mathtt{in}\ (\lambda x_1 x_2.x_2)(f\ \mathtt{1})(f\ \mathtt{true})$
with type `bool`, and this is also reduced to `true`. $\lambda\mu_{ml}$ could overcome the counterexample of polymorphic `callcc` in ML, and moreover, the typing conditions for let-expression could be deleted. In particular, $\lambda\mu_{iml}$ is another candidate for implicit polymorphism by value, compared with implicit polymorphism by name in Leroy [Lero93].

Ong&Stewart [OS97] extensively studied a call-by-value programming language based on a call-by-value variant of finitely typed $\lambda\mu$-calculus. There are some distinctions between Ong&Stewart and our finite type fragment; their reduction rules have type annotations like the complete Church-style, and, using the annotation, more reduction rules are defined than ours, which can give a stronger normal form. In addition, our notion of values is an extended one, which would be justified by observation based on the CPS-translation. Moreover, our renaming rule is applied for the extended values, and following the proof of lemma 4, this distinction is essential for the CPS-translation of renaming rule. Otherwise the reductions by renaming rule would not be simulated by β-reductions. On the other hand, in the equational theory λ_C of Hofmann [Hof95], one obtains $\alpha(\mathcal{C}(\lambda\beta.M)) =_{\mathcal{C}} M[\beta := \alpha]$ without restricting to values, which would be distinction between equational theory and rewriting theory.

We used the CPS-translation as a useful tool to show consistency and strong normalization of the system. With respect to Proposition 2 (soundness of CPS-translation); for call-by-name $\lambda\mu$, on the one hand, the completeness is obtained in de Groote [Groo94], i.e., the call-by-name CPS-translation is injective. For a call-by-value system with Felleisen's control operators [FFKD86], on the other hand, the completeness is established with respect to categorical models [Hof95], and moreover, this method is successfully applied to call-by-name $\lambda\mu$ [HS97]. We believe that our CPS-translation would be natural along the line of [Plot75], and it is worth pursuing the detailed relation to such categorical models [HS97, SR96].

Acknowledgements I am grateful to Susumu Hayashi, Yukiyoshi Kameyama, and the members of the Proof Animation Group for helpful discussions.

References

[Ando95] Y.Andou: A Normalization-Procedure for the First Order Classical Natural Deduction with Full Logical Symbols. Tsukuba Journal of Mathematics 19 (1) pp.153–162, 1995.

[Bare84] H.P.Barendregt: *The Lambda Calculus, Its Syntax and Semantics* (revised edition), North-Holland, 1984.

[Bare92] H.P.Barendregt: Lambda Calculi with Types, Handbook of Logic in Computer Science Vol.II, Oxford University Press, pp.1–189, 1992.

[BB93] F.Barbanera and S.Berardi: Extracting Constructive Context from Classical Logic via Control-like Reductions, Lecture Notes in Computer Science 664, pp.45–59, 1993.

[BS97] G.Barthe and M.H.Sørensen: Domain-free Pure Type Systems, Lecture Notes in Computer Science 1234, pp.9–20, 1997.

[DM82] L.Damas and R.Milner: Principal type-schemes for functional programs, *Proc. 9th Annual ACM Symposium on Principles of Programming Languages*, pp.207–212, 1982.

[Groo94] P.de Groote: A CPS-Translation for the $\lambda\mu$-Calculus, Lecture Notes in Computer Science 787, pp.85–99, 1994.

[Groo95] P.de Groote: A Simple Calculus of Exception Handling, Lecture Notes in Computer Science 902, pp.201–215, 1995.

[FFKD86] M.Felleisen, D.P.Friedman, E.Kohlbecker, and B.Duba: Reasoning with Continuations, *Proc. Annual IEEE Symposium on Logic in Computer Science*, pp.131–141, 1986.

[Fuji97] K.Fujita: Calculus of Classical Proofs I, Lecture Notes in Computer Science 1345, pp.321–335, 1997.

[Fuji98] K.Fujita: Polymorphic Call-by-Value Calculus based on Classical Proofs, Lecture Notes in Artificial Intelligence 1476, pp.170–182, 1998.

[Fuji99] K.Fujita: Explicitly Typed $\lambda\mu$-Calculus for Polymorphism and Call-by-Value, Lecture Notes in Computer Science 1581, pp.162–176, 1999.

[Grif90] T.G.Griffin: A Formulae-as-Types Notion of Control, *Proc. 17th Annual ACM Symposium on Principles of Programming Languages*, pp.47–58, 1990.

[HDM93] R.Harper, B.F.Duba, and D.MacQueen: Typing First-Class Continuations in ML, *J.Functional Programming*, 3 (4) pp.465–484, 1993.

[HL91] R.Harper and M.Lillibridge: ML with callcc is unsound, *The Types Form*, 8, July, 1991.

[HL93a] R.Harper and M.Lillibridge: Explicit polymorphism and CPS conversion, *Proc. 20th Annual ACM Symposium on Principles of Programming Languages*, pp.206–219, 1993.

[HL93b] R.Harper and M.Lillibridge: Polymorphic type assignment and CPS conversion, *LISP and Symbolic Computation* 6, pp.361–380, 1993.

[HM93] R.Harper and J.C.Mitchell: On The Type Structure of Standard ML, *ACM Transactions on Programming Languages and Systems*, Vol. 15, No.2, pp.210–252, 1993.

[Hof95] M.Hofmann: Sound and complete axiomatisations of call-by-value control operators, *Math.Struct. in Comp. Science* 5, pp.461–482, 1995.

[How80] W.Howard: *The Formulae-as-Types Notion of Constructions, To H.B.Curry: Essays on combinatory logic, lambda-calculus, and formalism*, Academic Press, pp.479–490, 1980.

[HS97] M.Hofmann and T.Streicher: Continuation models are universal for $\lambda\mu$-calculus, *Proc. 12th Annual IEEE Symposium on Logic in Computer Science*, 1997.

[Koba97] S.Kobayashi: Monads as modality, *Theor.Comput.Sci.* 175, pp.29–74, 1997.

[KTU94] A.J.Kfoury, J.Tiuryn, and P.Urzyczyn: An Analysis of ML Typability, *Journal of the Association for Computing Machinery*, Vol.41, No.2, pp.368–398, 1994.

[Lero93] X.Leroy: Polymorphism by name for references and continuations, *Proc. 20th Annual ACM Symposium of Principles of Programming Languages*, pp.220–231, 1993.

[Mit88] J.C.Mitchell: Polymorphic Type Inference and Containment, *Information and Computation* 76, pp.211–249, 1988.

[Mil78] R.Milner: A Theory of Type Polymorphism in Programming, *Journal of Computer and System Sciences* 17, pp.348–375, 1978.

[Murt91] C.R.Murthy: An Evaluation Semantics for Classical Proofs, *Proc. 6th Annual IEEE Symposium on Logic in Computer Science*, pp.96–107, 1991.

[MW85] A.Meyer and M.Wand: Continuation Semantics in Typed Lambda-Calculi, Lecture Notes in Computer Science 193, pp.219–224, 1985.

[Ong96] C.-H.L.Ong: A Semantic View of Classical Proofs: Type-Theoretic, Categorical, and Denotational Characterizations, *Linear Logic '96 Tokyo Meeting*, 1996.

[OS97] C.-H.L.Ong and C.A.Stewart: A Curry-Howard Foundation for Functional Computation with Control, *Proc. 24th Annual ACM Symposium of Principles of Programming Languages*, 1997.

[Pari92] M.Parigot: $\lambda\mu$-Calculus: An Algorithmic Interpretation of Classical Natural Deduction, Lecture Notes in Computer Science 624, pp.190–201, 1992.

[Pari93] M.Parigot: Classical Proofs as Programs, Lecture Notes in Computer Science 713, pp.263-276, 1993.

[Pari97] M.Parigot: Proofs of Strong Normalization for Second Order Classical Natural Deduction, J.Symbolic Logic 62 (4), pp.1461–1479, 1997.

[Plot75] G.Plotkin: Call-by-Name, Call-by-Value and the λ-Calculus, *Theor.Comput.Sci.* 1, pp. 125–159, 1975.

[Praw71] D.Prawitz: Ideas and Results in Proof Theory, *Proc. 2nd Scandinavian Logic Symposium*, edited by N.E.Fenstad, North-Holland, pp.235–307, 1971.

[RS94] N.J.Rehof and M.H.Sørensen: The λ_Δ-Calculus, Lecture Notes in Computer Science 789, pp.516–542, 1994.

[SR96] T.Streicher and B.Reus: Continuation semantics: abstract machines and control operators, to appear in *J.Functional Programming.*

[Taka89] M.Takahashi: Parallel Reductions in λ-Calculus, *J.Symbolic Computation* 7, pp.113–123, 1989.